Designing Exact Spaced Seed Filters Based on Combined Hit and Coverage Information

Moein Karami
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

Jens Zentgraf
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany
Saarbrücken Graduate School of Computer Science, SIC, Saarbrücken, Germany

Sven Rahmann[1]
Algorithmic Bioinformatics, Faculty of Mathematics and Computer Science,
Saarland University, Saarbrücken, Germany
Center for Bioinformatics Saar, Saarland Informatics Campus, Saarbrücken, Germany

Abstract

We revisit the classical problem of designing exact gapped k-mer based filtration methods to find all occurrences of a given query sequence (e.g., DNA read) in a text (genome) with at most a given number of substitutions. Whereas many existing filtration methods use small k and initiate a computationally expensive further investigation on a single k-mer hit to guarantee no false negatives, we derive stricter filtration criteria based on both the number of k-mer hits and hit-covered positions. Notably, our criteria go beyond a simple logical AND of hit-based and coverage-based criteria.

We provide methods based on both integer linear programs and dynamic programming to define optimal exact filter thresholds and compare the behavior of running times of both approaches. We then investigate to what degree a filter based on specific combinations of hits and coverage has better filtration efficiency than filters based on a single criterion (hits or coverage), or on a simple logical AND of both. We define two new quantities to characterize the filtration efficiency curve of a spaced seed for a specific sequence length and a desired tolerated number of changes. In a case study, we compare all symmetric masks with 25 significant positions in a window of 35 positions across four filtration criteria. Code is available at `https://gitlab.com/rahmannlab/seed-optimization`.

2012 ACM Subject Classification Applied computing → Molecular sequence analysis; Theory of computation → Discrete optimization; Applied computing → Bioinformatics

Keywords and phrases Spaced seed, Gapped k-mer, Hit, Coverage, Integer linear program (ILP), Dynamic programming (DP), Similarity search

Digital Object Identifier 10.4230/LIPIcs.WABI.2026.21

Supplementary Material *Software (Source code)*: `https://gitlab.com/rahmannlab/seed-optimization/` [9], archived at `swh:1:dir:050c8caccda68021f2feb9b7b8400444e9888340`

Acknowledgements We thank the four anonymous WABI reviewers whose comments have substantially improved the presentation of this work.

[1] Corresponding author

1 Introduction

Sequence similarity search is a fundamental problem in computational biology. Given a query sequence s of length n, a distance measure $d(\cdot, \cdot)$ on strings, and a tolerated threshold D on the distance, the task is to find all substrings t of a text T (e.g., a large database of sequences, concatenated by separators) that satisfy $d(s, t) \leq D$. For large enough n in relation to small enough D, the basic idea of efficient indexed search is the idea of *filtration*, i.e., that s and t must share a sufficiently large number of identical positions, which can be found quickly by looking up certain primitives in an index data structure. Early tools such as QUASAR [3] or SWIFT [14] used contiguous substrings of (relatively short) length (so called k-mers or q-grams, where k or q is the length, respectively). Around the same time, a line of work emerged that considered "gapped q-grams" or "spaced k-mers" that considered k (not necessarily consecutive) fixed positions in a window of length $w \geq k$, specified by a pattern κ called a *mask* or a (spaced) *seed*.

After their initial introduction by Burkhardt and Kärkkäinen [4], several studies have shown the superiority of spaced seeds over standard k-mer seeds, e.g. for genome alignment [6], for homology detection [1, 16], for metagenomic classification [5], or for phylogenetic tree reconstruction [8]. This has resulted in a large body of literature concerning designing *optimal* masks for given k and w and various specific objectives.

One has to distinguish the *probabilistic* setting, where sequence changes are introduced at every position with some probability [10, 13], independently of the other positions, from the *combinatorial* setting, where a fixed number of changes is considered. In the combinatorial setting that we consider here, the decision problem whether for a given mask κ, sequence length n, and number of changes c, it can be guaranteed that at least one spaced seed is unchanged (for all possible placements of the c changes) is NP-complete [11], but for reasonably short masks (small k, w), the problem, and related problems, can be solved using exponential-time dynamic programming algorithms over certain types of automata [2].

Recent work [15] presented integer linear program (ILP) formulations for two problems that go beyond the one-hit decision problem (see Section 2 for precise definitions of hits and hit-covered positions):

- MinHits(κ, n, c) asks for the minimum number of guaranteed "hits" (unchanged spaced seeds κ) when (at most) c positions in a sequence of length n are changed, minimized over all possible placements of the changes.
- MinCov(κ, n, c) asks for the minimum number of guaranteed hit-covered positions under the same circumstances, where a position is covered if it is any of the k unchanged positions of a hit.

An illustration of hits and covered positions is shown in Figure 1. Precise definitions are given in Section 2. Computing these properties of all masks sharing common parameters (k, w) (or of a restricted subset, such as symmetric ones) allows us to compare the masks in this class and list the "best" ones. Tables of best-in-class masks (maximizing MinHits or MinCov over all symmetric masks with fixed k, w) and their properties are available [15].

While the MinHits or MinCov value can be considered a natural quality measure of a mask, it does not have a direct interpretation: It does not tell us, for example, which fraction of sequences of length n with more than c changes, say $2c$ changes, achieve such a number of hits or such a coverage. However, such an analysis is relevant to understand how a filter designed based on hits or coverage would perform on sequences that are less similar than the filter design specified.

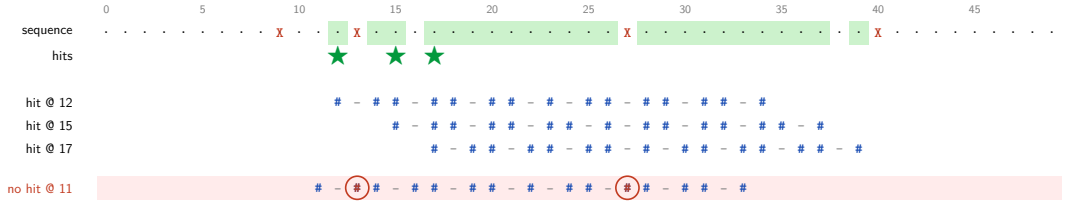

Figure 1 A sequence of length $n = 50$ with $c = 4$ changes at positions $\{9, 13, 27, 40\}$ (red `X`s), with starting positions of all 3 hits (green stars at positions 12, 15, 17) and the thereby covered 24 positions (light green shaded sequence positions) when using the $(15, 23)$-shaped mask `#_##_##_##_#_##_##_##_#`. In addition, a non-hit at starting position 11 is shown, where two of the significant positions lie on changed sequence positions (13 and 27), illustrated by red circled `#`s.

Here, we extend previous work into several directions, specifically with a similarity search filter perspective in mind.

1. We examine filtration efficiency curves. Given a mask κ, a sequence length n, and a filtering condition F involving hits, coverage, or combinations thereof, we define $\textsc{Filtered}_F(\kappa, n, c)$ as the fraction of sequences of length n with exactly c changes (among all $\binom{n}{c}$ such sequences) that do not satisfy condition F. For example, we may take F as the MinHits filter by specifying the condition that the number of observed hits must be at least $\textsc{MinHits}(\kappa, n, c^*)$ for a chosen c^*. Then by design, $\textsc{Filtered}_F(\kappa, n, c) = 0$ for all $c \leq c^*$, but we are interested in how fast the function $c \mapsto \textsc{Filtered}_F(\kappa, n, c)$ grows for $c > c^*$. This function is the filtration efficiency curve. As an illustration, Figure 2 shows four examples of such curves for $c^* = 3$, $n = 100$ for two distinct masks with $k = 21$ and $w = 27$, taking F as the MinHits and MinCov filter, respectively.
2. We improve filtration efficiency by considering not hits or coverage alone, but combinations of both. An obvious and simple idea is to take the logical AND of the two conditions that observed hits are at least MinHits and observed coverage is at least MinCov. However, as might be expected, this does not improve filtration efficiency by much. A new insight of this work is that, conditioned on a fixed number of hits h, the MinCov value may change in a *non-monotone* way. We therefore show how to compute hit-specific minimum coverage: Let $\textsc{MinCov}|\textsc{Hits}(\kappa, n, c, h)$ be the minimum number of covered positions in a sequence of length n with at most c changes that result in exactly h hits. We present an integer linear program that is an extension of the MinCov ILP from [15].
3. Complementary to the ILP approach, we provide dynamic programming solutions for the computational problems described. While a basic DP strategy is relatively straightforward, we point out several optimizations to reduce computation time.

While the idea to filter by coverage is not new [4, 12], we are not aware of previous work that combines hit count and coverage to improve filtration efficiency.

We proceed as follows. In Section 2, we give necessary definitions for the remainder of this work. In Section 3, we collect the relevant computational problems and develop methods for solving them. Section 4 is devoted to experimental evaluations: In Section 4.1, we compare running times of different approaches for solving the optimization problems, and in Section 4.2, we compare filtration efficiency curves of different masks and different filter types. Furthermore, we introduce new properties of masks/filter combinations related to the location and steepness of the filtration curve. We conclude with a discussion in Section 5.

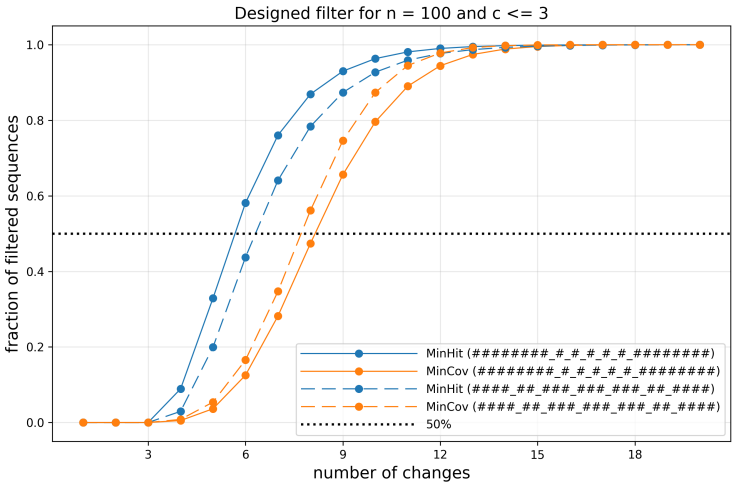

Figure 2 Filtration curves for two $(k, w) = (21, 27)$ masks μ^1 = `########_#_#_#_#_#_########` (solid lines) and μ^2 = `####_##_###_###_###_##_####`, with filtration criteria designed for sequence length $n = 100$ and at most 3 changes, based on hits alone (blue color; $\textsc{MinHits}(\mu^1, 100, 3) = 18$, $\textsc{MinHits}(\mu^2, 100, 3) = 15$) and on coverage alone (orange color; $\textsc{MinCov}(\mu^1, 100, 3) = 57$, $\textsc{MinCov}(\mu^2, 100, 3) = 66$).

2 Preliminaries

We begin by introducing basic notation, masks and their representations, κ-mers of a sequence as generalizations of k-mers. Then we explain the effect of single-position changes on the conservation of κ-mers and define *hits* and *covered positions*. Most of these preliminaries are the same as in our previous work [15]. Then, we introduce filtration strategies based on κ-mers and filtration efficiency curves.

Basic definitions. We write $[n]$ for the set of integers $\{0, \ldots, n-1\}$. Indexing of strings starts at 0, so $s = (s_0, \ldots, s_{n-1}) = (s_i)_{i \in [n]}$, where the length of s is written as $|s| = n$.

We define the *cumulative binomial coefficient* $\binom{w}{\leq c} := \binom{w}{c} + \binom{w}{c-1} + \cdots + \binom{w}{0}$ for integers $w \geq c \geq 0$; it counts the number of subsets of $[w]$ of size at most c.

Masks of shape (k, w). Given two integers $w \geq k \geq 2$, a (k, w)-*mask* is a string μ of length w over the alphabet $\{\texttt{\#}, \texttt{\_}\}$ that contains exactly k times the character `#` and $w - k$ times the character `_`. The positions marked `#` are called *significant*, and the positions marked `_` are called *insignificant*, jokers, "don't care" positions, spaces or gaps. We call k the *weight* of the mask and w its *width* or *window length*. The pair (k, w) is called the *shape* of the mask. A mask μ may also be represented as the tuple κ of significant positions: $\kappa = \{j : 0 \leq j < w \text{ and } \mu_j = \texttt{\#}\}$.

We require that $\mu_0 = \mu_{w-1} = \texttt{\#}$, because otherwise the mask could be shortened by removing the insignificant characters at each end to obtain an equivalent smaller mask (disregarding matches at sequence boundaries). With this constraint, there are $\binom{w-2}{k-2}$ different masks of shape (k, w).

A mask μ is *symmetric* if $\mu = rev(\mu) := \mu_{w-1} \ldots \mu_1 \mu_0$. We consider only symmetric masks in this work because symmetry removes the need to separately consider both forward and reverse strand of a DNA molecule; the mask is the same on both strands. This allows working

with canonical gapped k-mers that represent both the forward and the reverse complementary k-mers with the same significant positions in the same window. We also restrict ourselves to odd k to avoid any issues with k-mers that are equal to their reverse complement. We write $\mathcal{M}(k,w)$ for the set of all mask tuples κ (with significant ends) of shape (k,w) and $\mathcal{S}(k,w)$ for the subset of all symmetric ones. For example, $\kappa = (0,2,5,8,10) \in \mathcal{S}(5,11)$, corresponding to $\mu =$ `#_#__#__#_#`. For $k = 25$ and $w = 35$, we find $|\mathcal{M}(k,w)| = 92\,561\,040$ and $|\mathcal{S}(k,w)| = 4\,368$.

κ-mers of a sequence and effects of changes. Given a mask μ of shape (k,w) with its corresponding offset tuple κ and a string s of length n over an alphabet (e.g., the DNA or protein alphabet), we obtain $n-w+1$ *spaced k-mers* from s: The i-th spaced k-mer g_i, for $i \in [n-w+1]$, is obtained by concatenating the significant positions when the mask is applied at positions $i,\ldots,i+w-1$ of s, i.e., $g_i = (s_{i+j})_{j\in\kappa}$. We also say that each such string g_i is a *κ-mer* of s, referring to the mask's tuple representation κ.

When we change a single character in a sequence s, then up to k of the κ-mers may be changed (less if the changed position is near an end of the sequence). We say that the κ-mer starting at position $p \in [n-w+1]$ (κ-mer number p for short) is changed or destroyed if any of the positions $\{p+j : j \in \kappa\}$ is changed. Otherwise, the κ-mer is unchanged or unaffected. In this case, we also say that the κ-mer is a *hit* to the sequence.

We say that sequence position $i \in [n]$ is *covered* if there exists an unaffected κ-mer (hit) starting at some position $p \le i$ such that $i = p + j$ for some $j \in \kappa$. In other words, if the κ-mer starting at position p is a hit, then all positions $p+j$ for $j \in \kappa$ are covered.

Hits and covered positions are illustrated in Figure 1. We emphasize that the definitions are based on κ-mer identity by position, not sequence content, which makes them independent of any alphabet. This allows us to model sequences with changes as binary sequences $x = (x_i)$ over $\{0,1\}$ with 0 indicating an unchanged position and 1 indicating a changed position, such that $c = \sum_i x_i$ is the number of changes.

Hit-based and coverage-based filters for similarity search. A classical application of k-mer (and κ-mer) based index data structures is to construct filtration strategies for efficient similarity search. Given a query sequence of length n, one wants to find all substrings of a long (database) sequence that differ from the query in at most c positions (Hamming distance $\le c$), where c is a user-defined parameter. In practice, of course, one would be interested in (weighted) edit distance; and gaps may be accommodated by not looking at database substrings of length exactly n but of larger lengths $n^+ > n$. Some implementations even use much larger blocks of the database to simplify bookkeeping at the expense of a slightly increased chance of random hits (and arbitrary positions in the block).

Any sufficiently similar sequence of length n (or longer) must share at least a certain number of unchanged κ-mers with the query; and we call this number $\textsc{MinHits}(\kappa,n,c)$. More formally, we define $\textsc{MinHits}(\kappa,n,c)$ as the smallest possible number of unchanged κ-mers in a sequence of length n when at most c arbitrary positions are changed, i.e., the minimum over all $\binom{n}{\le c}$ arrangements of changed positions. Of course, such a filtration strategy is only meaningful for the given parameters if $\textsc{MinHits}(\kappa,n,c) > 0$, and ideally if it is a large number, such that it is highly improbable that "random" sequences achieve that many hits.

Similarly, one may consider a coverage-based filtration strategy and ask for $\textsc{MinCov}(\kappa,n,c)$, defined as the minimum possible number of covered positions in a sequence of length n when at most c positions are changed.

Our earlier work [15] introduced these optimization problems, provided integer linear programs (ILPs) for solving them and compared all symmetric masks of different shapes (k, w) with the idea that a larger MinHits value is better than a smaller one, and that a larger MinCov value is better than a smaller one, because random sequences are less likely to achieve them. However, we did *not* consider or compare the filtration efficiency $\text{Filtered}_F(\kappa, n, c)$, defined as the fraction of the $\binom{n}{c}$ sequences of length n with c changes that do not pass the filter criteria F with mask κ. We may express

$$\text{Filtered}_F(\kappa, n, c) = 1 - \frac{\text{Count}_F(\kappa, n, c)}{\binom{n}{c}},$$

where $\text{Count}_F(\kappa, n, c)$ is the number of sequences of length n with exactly c changes that do pass the condition F with mask κ. We say that a filter (condition) F is *exact for* or *designed for* mask κ and c^* changes among n positions if $\text{Filtered}_F(\kappa, n, c) = 0$ for all $c \leq c^*$.

3 Computational Problems and Algorithms for Exact Filters

Given a mask κ (we typically use the tuple form κ, but may use the string form μ interchangeably) of shape (k, w), a sequence length n, and a number of changes c satisfying $0 \leq c \leq n$ and $0 \leq k \leq w \leq n$, we consider the following computational problems about optimal placement of at most c changes in a sequence of length n.

- Compute $\text{MaxChg}(\kappa, n)$, the maximal number of changes such that, irrespective of their placement, there is at least one hit.
- Compute $\text{MinHits}(\kappa, n, c)$, the smallest possible number of hits for any placement of the c changes (or fewer). For $c > \text{MaxChg}(\kappa, n)$, we have $\text{MinHits}(\kappa, n, c) = 0$ by definition, so the problem is only of interest for $c \leq \text{MaxChg}(\kappa, n)$.
- Compute $\text{MinCov}(\kappa, n, c)$, the smallest possible number of covered positions for any placement of the c changes (or fewer). Again, we may assume $c \leq \text{MaxChg}(\kappa, n)$.
- Compute $\text{MinCov}|\text{Hits}(\kappa, n, c, h)$, the smallest possible number of covered positions for any placement of *at most* c changes under the condition that there are *exactly* h hits. The value is undefined for $h < \text{MinHits}(\kappa, n, c)$, and we set it to ∞. This yields a table $h \mapsto \text{MinCov}|\text{Hits}(\kappa, n, c, h)$, which we refer to as $\text{MinCov}|\text{Hits}(\kappa, n, c)$, or simply as the "MinCov table". It holds that $\text{MinCov}(\kappa, n, c) = \min_{h=0,\ldots,n-w+1} \text{MinCov}|\text{Hits}(\kappa, n, c, h)$. This problem has not been considered previously.
- For a specified filter condition F, compute $\text{Count}_F(\kappa, n, c)$, the number of sequences of length n with exactly c changes that satisfy condition F with mask κ. The map $c \mapsto \text{Filtered}_F(\kappa, n, c) = 1 - \text{Count}_F(\kappa, n, c)/\binom{n}{c}$ is the filtration efficiency curve for the given filter and parameters. Such problems have mostly been considered in the probabilistic setting for F being the condition that the number of hits exceeds a threshold (often 1). We consider more complex filter conditions, as explained below.

We note the equivalence $\text{MinHits}(\kappa, n, c^*) = 0 \leftrightarrow \text{MinCov}(\kappa, n, c^*) = 0$ (such filters are useless), and also $\text{MinHits}(\kappa, n, c^*) = 1 \leftrightarrow \text{MinCov}(\kappa, n, c^*) = k$, as well as $\text{MinHits}(\kappa, n, c^*) > 1 \leftrightarrow \text{MinCov}(\kappa, n, c^*) > k$.

From solutions to these computational problems, we may design different exact filters (and find their filtration efficiency curves) as follows. We assume that κ, n and c^* are given, and we want to design filter conditions that are exact for $c \leq c^*$ changes.

MinHits filter: Compute $h^* := \text{MinHits}(\kappa, n, c^*)$. When scanning the $n - w + 1$ κ-mers of a query sequence of length n, count the number of hits h (to an indexed sequence). Define F as the condition $h \geq h^*$, thereby ensuring $\text{Filtered}_F(\kappa, n, c) = 0$ for $c \leq c^*$.

Table 1 Integer linear program for solving MinCov|Hits(κ, n, c, h) for a (k, w)-shaped mask κ. The k-tuple κ contains the indices $j \in [w]$ of the significant positions of the mask. All variables are integer binary, x_i $(i \in [n])$, y_p $(p \in [n-w+1])$ and z_i $(i \in [n])$, as explained in the text.

$$\text{Minimize} \quad \sum_{i\in[n]} z_i\,, \tag{1}$$

$$\text{such that} \quad \sum_{i\in[n]} x_i \le c\,, \tag{2}$$

$$y_p \ge 1 - \sum_{j\in\kappa} x_{p+j} \qquad \text{for all } p \in [n-w+1]\,, \tag{3}$$

$$y_p \le 1 - x_{p+j} \qquad \text{for all } p \in [n-w+1],\ j \in \kappa\,, \tag{4}$$

$$\sum_{p\in[n-w+1]} y_p = h\,, \tag{5}$$

$$z_{p+j} \ge y_p \qquad \text{for all } p \in [n-w+1],\ j \in \kappa\,, \tag{6}$$

$$z_i \le \sum_{j\in\kappa,\ i-j\ge 0,\ i-j\le n-w} y_{i-j} \qquad \text{for all } i \in [n]\,, \tag{7}$$

$$\text{and} \quad x_i \in \{0,1\},\ y_p \in \{0,1\},\ z_i \in \{0,1\} \qquad \text{for all } i \in [n],\ p \in [n-w+1]\,.$$

MinCov filter: Compute $\gamma^* := \text{MinCov}(\kappa, n, c^*)$. When scanning the κ-mers of the query, determine the number of hit-covered positions γ (this requires more work than just counting hits h). Define F as the condition $\gamma \ge \gamma^*$, ensuring the exactness condition $\text{Filtered}_F(\kappa, n, c) = 0$ for all $c \le c^*$.

AND-combined filter: Compute both h^* and γ^* as above. Determine both h and γ of the query sequence. (The computation of h is essentially free when computing γ.) The filter condition F becomes "$h \ge h^*$ and $\gamma \ge \gamma^*$", which is exact.

MinCov table filter: Compute the table $G^*[h] := \text{MinCov|Hits}(\kappa, n, c^*, h)$ for all $h = 0, \ldots, n-w+1$. As for the AND-combined filter, determine both h and γ of the query sequence. Now the filter condition F is $\gamma \ge G^*[h]$, which is exact by construction.

In Section 4.2, we compare filtration efficiency curves of these four filter types for the same mask.

For now, we are interested in algorithms for computing the relevant quantities. We first present an ILP for computing the MinCov table $G^*[h]$ by iterating over all hit numbers $h \in [n-w+1]$; see Section 3.1. Then we give dynamic programming algorithms for MinHits, MinCov, and MinCov|Hits in Sections 3.2, 3.3, 3.4, respectively. Finally, we also give a DP algorithm for Count (Section 3.5).

3.1 Integer linear program for MinCov|Hits

Previous work [15] presented ILPs for MaxChg, MinHits and MinCov. Here, we give a direct extension of the MinCov ILP for MinCov|Hits. We use the following *binary* variables with the described semantics.

- $x = (x_i)_{i\in[n]}$, indicators of changed positions: $x_i = 1$ if and only if sequence position i is changed;
- $y = (y_p)_{p\in[n-w+1]}$, indicators of (starting positions of) unchanged κ-mers: $y_p = 1$ if and only if the κ-mer starting at position p is unchanged (i.e., none of its significant positions is changed),
- $z = (z_i)_{i\in[n]}$, indicators of covered positions: $z_i = 1$ if and only if sequence position i is covered by an unchanged κ-mer.

Table 2 Examples for MinCov|Hits$(\kappa, 100, 3, h)$ values for different masks κ and $h \in \{4, \ldots, 18\}$. While the sequence increases regularly by 1 for the standard 25-mer mask, it is non-monotone for the two (25,35) masks shown here. Note the striking drop to 60 and 54 for 17 and 18 hits, respectively, for the third mask. Numbers not shown are undefined (e.g., because MinHits$(\kappa, 100, 3) = 10$ for the third mask).

MinCov|Hits$(\kappa(\mu), 100, 3, h)$ values	Hit count h														
mask μ ($k = 25$, $w \in \{25, 35\}$)	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
`#########################`	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42
`#__##_###__#############__###_##__#`		41	43	44	46	47	48	49	50	51	52	51	52	53	52
`###_#####__##_##_#_##_##__#####_###`							70	73	74	73	74	73	74	60	54

We use $i \in [n]$ for indexing sequence positions, $p \in [n - w + 1]$ for indexing starting positions of κ-mers, and j for an element of κ.

▷ Claim 1. The ILP in Table 1 solves MinCov|Hits(κ, n, c, h).

Proof idea. The objective (1) minimizes the number of covered positions, as intended. Inequality (2) ensures that we use at most c changes. Inequalities (3) ensure that $y_p = 1$ if all positions of the κ-mer starting at p are unchanged, and (4) conversely ensures that $y_p = 0$ if any position of that κ-mer is changed. Equation (5) ensures that we have exactly h hits. Inequalities (6) force the z-variables of all significant positions covered by a hit to 1, whereas (7) forces the z-variables of positions not covered by any such κ-mer to zero. The last group of inequalities (7) may be omitted here because the objective minimizes the z_i anyway. ◁

Examples of MinCov tables are shown in Table 2. It is noteworthy and perhaps unexpected that the function $h \mapsto$ MinCov|Hits(κ, n, c, h) is non-monotone for some masks κ (and for given n, c), so the overall MinCov value is not achieved for $h =$ MinHits(κ, n, c), but for a strictly larger value.

3.2 Dynamic programming solution of MinHits

For what follows, fix the mask κ. We tabulate the solution to MinHits(κ, i, j) for increasing sequence length $i = w, \ldots, n$ (starting with a single window of length w) and increasing number of changes $j = 0, \ldots, c$, remembering the relative positions of the changes in the current window (as a so-called *change history set* Γ) and the total number of changes used. Concretely, for $i = w, \ldots, n$, $j = 0, \ldots, c$ and all subsets Γ of $[w]$ of size at most c, let $H[i, \Gamma, j]$ be the minimum number of hits in a sequence of length i (with positions numbered $0, \ldots, i - 1$), having used $0 \le j \le c$ changes in these i positions, where the most recent changes have occurred at positions $i - w + q$, $q \in \Gamma$, satisfying $0 \le q < w$. There are $\binom{w}{\le c} \in O(w^c)$ sets Γ to be considered, which is smaller than $\binom{n}{c}$, the number of change placements over a length-n sequence. The solution of MinHits(κ, n, c) is then given by $\min_\Gamma H[n, \Gamma, c]$.

Initialization for length $i = w$ is straightforward:

$$H[w, \Gamma, j] = \begin{cases} \infty & \text{if } |\Gamma| \neq j \quad \text{(invalid combination of \#changes } j \text{ and their positions } \Gamma\text{)}, \\ 1 & \text{if } |\Gamma| = j \text{ and } \kappa \cap \Gamma = \{\} \quad \text{(all changes in non-significant positions)}, \\ 0 & \text{if } |\Gamma| = j \text{ and } \kappa \cap \Gamma \neq \{\} \quad \text{(a change in a significant position)}. \end{cases}$$

We now describe how $H^+ := H[i + 1, \cdot, \cdot]$ is computed from $H^0 := H[i, \cdot, \cdot]$ for $i = w, \ldots, n - 1$. Older values $H[i', \cdot, \cdot]$ for $i' < i$ are not required.

While it is customary to describe the new value in terms of the old values, e.g. $H^+[\Gamma, j] = \min\{\dots\}$, the implementation becomes simpler when we initialize the new values $H^+[\Gamma, j] \leftarrow \infty$ and then iterate over the existing values $H^0[\Gamma, j]$ and update the dependent H^+ values (forward dynamic programming). For this purpose, we define the $\xleftarrow{}_{\min}$ operator: $x \xleftarrow{}_{\min} y$ means $x \leftarrow \min\{x, y\}$.

The main decision to make when elongating the sequence by one character is whether to place a change at the new position (numbered i) or not, so we consider both cases. A change can only be placed if we have not used all c changes yet. When moving by one position, the relative positions of the existing changes in Γ decrease by 1, and resulting negative positions are removed. Thus, we define $\Gamma \ominus 1 := \{p - 1 \mid p \in \Gamma \text{ and } p \geq 1\}$.

If a new change is introduced at the new position, the number of used changes increases by 1 and $w - 1$ becomes included in Γ. (This is only possible if $|\Gamma|$ does not become larger than c by this addition.) In this case, the newly generated κ-mer cannot be a hit because the new change occurs at the significant position $(w - 1) \in \kappa$. If no new change is introduced, we have to check whether one of the shifted existing changes covers a significant position of κ. A new hit is obtained if and only if this is not the case. We thus write $\delta_\Gamma := 1$ if and only if $(\Gamma \ominus 1) \cap \kappa = \{\}$, and $\delta_\Gamma := 0$ otherwise.

In summary, the value $h := H^0[\Gamma, j]$ is used to update (at most) two values of H^+:

1. $H^+[\Gamma \ominus 1,\ j] \ \xleftarrow{}_{\min}\ (h + \delta_\Gamma)$ (without using a change at the new position),
2. If $j < c$, then $H^+[(\Gamma \ominus 1) \cup \{w - 1\},\ j + 1] \ \xleftarrow{}_{\min}\ h$ (using a change at the new position; note that the condition $j < c$ implies that $|\Gamma| < c$).

An efficient implementation uses an (arbitrary) integer-encoding of the $\binom{w}{\leq c}$ sets Γ and a precomputed array that maps Γ to δ_Γ and its two possible successors $\Gamma \ominus 1$ and $(\Gamma \ominus 1) \cup \{w-1\}$, which permits a rapid inner loop implementation.

▷ Claim 2. The MinHits DP algorithm needs time $O(nc \cdot \binom{w}{\leq c})$ and space $O(c \cdot \binom{w}{\leq c})$.

Proof. $H[i, \Gamma, j]$ is computed by one initialization and at most two updates, for $(n - w + 1) \in O(n)$ values of i, at most $\binom{w}{\leq c}$ sets Γ (subsets of $[w]$ of size at most c) for $(c + 1) \in O(c)$ values of j. During the computation, only the H-values for the current value of i and the next value $i + 1$ need to be in memory; these are two arrays of size $\binom{w}{\leq c} \cdot (c + 1)$. ◁

3.3 Dynamic programming solution of MinCov

We proceed similarly to the MinHits solution, but we must consider a larger context of recent changes. This larger context is necessary to know which positions are newly covered and which positions were already covered when a new hit occurs at the next position.

We define $C[i, \Delta, j]$ as the minimum number of covered positions in a sequence of length $i \geq w$, having already used $0 \leq j \leq c$ changes in these first i sequence positions $0, \dots, i - 1$, where the most recent changes have occurred at positions $i - w + q$, with $q \in \Delta$ satisfying $-w + 1 \leq q \leq w - 1$. So Δ is a subset of $\{-w + 1, \dots, 0, \dots, w - 1\}$ of size at most c, and there are $\binom{2w-1}{\leq c}$ such sets. By $\Delta \geq 0$, we mean that all elements of Δ are non-negative.

Initialization for length $i = w$ is similar to MinHits:

$$C[w, \Delta, j] = \begin{cases} k & \text{if } \Delta \geq 0,\ |\Delta| = j,\ \text{and } \kappa \cap \Delta = \{\} \quad \text{(hit, } k \text{ positions covered)}, \\ 0 & \text{if } \Delta \geq 0,\ |\Delta| = j,\ \text{and } \kappa \cap \Delta \neq \{\} \quad \text{(not a hit)}, \\ \infty & \text{if } \Delta \not\geq 0 \text{ or } |\Delta| \neq j \quad \text{(invalid combination)}. \end{cases}$$

Note that values of C can be 0 or $\geq k$, but not in $\{1, \dots, k - 1\}$ because the first hit immediately covers k positions.

We now describe how $C^+ := C[i+1,\cdot,\cdot]$ is computed from $C^0 := C[i,\cdot,\cdot]$ for $i = w,\ldots,n-1$. Earlier values $C[i',\cdot,\cdot]$ for $i' < i$ are not required.

We define $\Delta \boxminus 1 := \{p-1 \mid p \in \Delta \text{ and } p \geq -w+2\}$. We write A_Δ for the number of additional positions that will be covered if a hit occurs at the new position, and set $A_\Delta = 0$ if no hit occurs. This number can be computed from the positions stored in Δ (in fact, in Δ^+, which is a function of Δ); details are given below.

We use $\gamma := C^0[\Delta, j]$ to update (at most) two values in C^+ using the $\leftarrow_{\overline{\min}}$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $C^+[\Delta^+,\, j] \;\leftarrow_{\overline{\min}}\; (\gamma + A_\Delta)$,
   where A_Δ is the (precomputed) number of newly covered positions in this configuration.
2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions. (Recall that the last position $w-1$ of the mask κ is significant.)
   If $j < c$ and $|\Delta^\dagger| \leq c$, then we update $C^+[\Delta^\dagger,\, j+1] \;\leftarrow_{\overline{\min}}\; \gamma$.

To determine A_Δ when we do not introduce a change at sequence position i, we need to determine first whether we have a hit or not (which works as for MinHits), but then also which positions in $\{i-w+q : q = 0,\ldots,w-1\}$ were previously covered, so we do not count them again. The latter is completely determined by the changes in the range $\{i-w+q : q = -w+1,\ldots,0,\ldots,w-1\}$, i.e., by Δ^+, which is determined by Δ. In an implementation, the Δ sets are integer-encoded and the A_Δ-values are pre-computed and tabulated.

▷ Claim 3. The MinCov DP algorithm needs time $O(nc \cdot \binom{2w-1}{\leq c})$ and space $O(c \cdot \binom{2w-1}{\leq c})$.

Proof. Similar to the proof of Claim 2. ◁

3.4 Dynamic programming solution of MinCov|Hits

The MinCov DP algorithm can be extended to also keep track the number of hits seen so far with essentially the same logic but an extra dimension of the table. Let $1 \leq \bar{h} \leq n-w+1$ be the upper limit on the number of hits to keep track of. We distinguish the $\bar{h}+2$ values $0,1,\ldots,\bar{h},\bar{h}+1$, where $\bar{h}+1$ represents all values $h > \bar{h}$.

They key quantity to tabulate is $M[i,\Delta,j,h]$, defined as the minimum number of covered positions in a sequence of length $i \geq w$, having used $0 \leq j \leq c$ changes and had $0 \leq h \leq \bar{h}$ hits in these first i sequence positions $0,\ldots,i-1$, where the most recent changes have occurred at positions $i-w+q$, with $q \in \Delta$ satisfying $-w+1 \leq q \leq w-1$. (For $h = \bar{h}+1$, the semantics are slightly different: … and had more than $\bar{h}$ hits …)

Initialization follows the MinCov logic, additionally counting hits:

$$M[w,\Delta,j,h] = \begin{cases} k & \text{if } h=1,\ \Delta \geq 0,\ |\Delta| = j \text{ and } \kappa \cap \Delta = \{\} \quad \text{(hit, } k \text{ positions covered)}, \\ 0 & \text{if } h=0,\ \Delta \geq 0,\ |\Delta| = j \text{ and } \kappa \cap \Delta \neq \{\} \quad \text{(not a hit)}, \\ \infty & \text{otherwise.} \end{cases}$$

With analogous notation as in Section 3.3 for MinCov, we use $m := M^0[\Delta,j,h]$ to update (at most) two values in M^+ using the $\leftarrow_{\overline{\min}}$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $M^+[\Delta^+,\, j, h^+] \;\leftarrow_{\overline{\min}}\; (m + A_\Delta)$,
   where A_Δ is the (precomputed) number of newly covered positions in this configuration, and $h^+ = \min(h+1, \bar{h}+1)$ if we have a hit, but $h^+ = h$ if not (we also pre-compute from Δ if there is a hit or not).

2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions.
   If $j < c$ and $|\Delta^\dagger| \le c$, then we update $M^+[\Delta^\dagger,\, j+1, h] \;\leftarrow_{\min}\; m$.

▷ Claim 4. The MinCov|Hits DP algorithm needs time $O(n\bar{h}c \cdot \binom{2w-1}{\le c})$ and space $O(\bar{h}c \cdot \binom{2w-1}{\le c})$.

Proof. Similar to the proof of Claim 2, taking into account the additional dimension. Note that $\bar{h} \le (n-w+1) \in O(n)$. ◁

3.5 Dynamic programming solution of Count

In principle, we may adapt the same logic as in the previous sections for solving $\textsc{Count}_F(\kappa, n, c)$. To handle all introduced filters F, including the MinCov table filter, we need to compute values $Y[i, \Delta, c, h, \gamma]$, defined as the *number of sequences* of length $i \ge w$, with $0 \le c \le i$ changes, $0 \le h \le \bar{h}$ hits and γ covered positions, where the most recent changes have occurred at positions $i-w+q$, with $q \in \Delta$ satisfying $-w+1 \le q \le w-1$. (For the MinHits filter, no coverage and less context is needed and things can be simplified.)

We only outline the method, because we will immediately argue that it is not practical. Instead of the $\leftarrow_{\min}$ operation, we need the $\leftarrow_+$ operation, which is the familiar in-place addition operator `+=` from C++ or Python. The change history sets Δ range over *all* $\binom{2w-1}{\le 2w-1} = 2^{2w-1}$ subsets of $\{-w+1, \ldots, w-1\}$ because the number of changes is not limited except by the sequence length n. For $w = 35$, these are in principle 2^{69} sets to consider, which is infeasible given standard time and memory resources. Nonetheless, by limiting w or the filtration efficiency curve to small numbers of changes c, the following procedure may in principle be used.

For sequence length $i = w$, we must have $c \le w$ and the change history Δ completely determines the sequence. Therefore,

$$Y[w, \Delta, c, h, \gamma] = \begin{cases} 1 & \text{if } h=1,\ \gamma=k,\ \Delta \ge 0,\ |\Delta|=c \text{ and } \kappa \cap \Delta = \{\} \quad \text{(hit)}, \\ 1 & \text{if } h=0,\ \gamma=0,\ \Delta \ge 0,\ |\Delta|=c \text{ and } \kappa \cap \Delta \neq \{\} \quad \text{(no hit)}, \\ 0 & \text{otherwise (impossible combination)}. \end{cases}$$

There are $\binom{w-k}{c}$ sets Δ that are hits ($h=1$, $\gamma=k$), where all changes are among the $w-k$ positions outside κ, and $\binom{w}{c} - \binom{w-k}{c}$ sets Δ that are non-hits ($h=0$, $\gamma=0$).

With analogous notation (Y^0, Y^+) as previously, we use $y := Y^0[\Delta, c, h, \gamma]$ to update *exactly* two values in Y^+ using the $\leftarrow_+$ operation:

1. Let $\Delta^+ := (\Delta \boxminus 1)$; we do not use a change at the new position.
   We update $Y^+[\Delta^+,\, c,\, h^+,\, \gamma + A_\Delta] \;\leftarrow_+\; y$, where A_Δ and h^+ are as in Section 3.4.
2. Let $\Delta^\dagger := (\Delta \boxminus 1) \cup \{w-1\}$; we use a change at the new position, so there can be no hit or newly covered positions. We update $Y^+[\Delta^\dagger,\, c+1, h, \gamma] \;\leftarrow_+\; y$.

As each count y from Y^0 is added twice to counts in Y^+, the number of sequences doubles when the sequence length increases by 1, and since the sum over $Y[w, \ldots]$ is initially $\sum_{c=0}^{w} \binom{w}{c} = 2^w$, the invariant that the sum over $Y[i, \ldots]$ is 2^i is maintained for all $i \ge w$.

▷ Claim 5. The Count DP algorithm needs time $O(n^4 \cdot 2^{2w-1})$ and space $O(n^3 \cdot 2^{2w-1})$.

Proof. Each of the four i, c, h, γ indices into Y may take $O(n)$ distinct values; there are 2^{2w-1} sets Δ to consider, as c is not bounded. Updates are constant-time per table value, and only the current $Y[i, \ldots]$ and $Y[i+1, \ldots]$ are in memory at the same time. ◁

3.6 Optimizations and Practical Considerations

We discuss improvements and practical alternatives to the basic DP algorithms.

The counting DP and filtration efficiency curves. Claim 5 shows that it is often impractical to use exact counting via dynamic programming to compute filtration efficiency curves, such as those in Figure 2. On the other hand, the filtration efficiencies do not need to be known exactly. Therefore, we may fall back to random sampling (with 10 million samples) for each value of c to determine $\textsc{Filtered}_F(\kappa, n, c)$ up to 3 digits. Whenever $\binom{n}{c} \leq 10^7$, we can compute $\textsc{Filtered}_F(\kappa, n, c)$ almost instantly exactly by exhaustive enumeration of all sequences and computing the resulting number of κ-hits and covered positions.

There may also exist more efficient methods to compute $\textsc{Count}_F(\kappa, n, c)$ exactly, based on a more refined automaton-based analysis of the language of binary sequences to be enumerated [2], but the general problem of large state space appears unavoidable.

3.6.1 Engineering Optimizations

Halving the sequence length. We observe that the minimization DPs need not be run until $i = n$ but only up to approximately $n/2$. We discuss the optimization for the (simpler) MinHits case. For the MinCov case, the idea is similar, but we need a history of size of $2w - 1$ (Δ in Sections 3.3 and 3.4) instead of a size-w history (Γ in Section 3.2. For the MinCov|Hits case, we additionally need to track hits.)

Consider a history window Γ of length w placed near the middle of the sequence. Suppose this window contains $c' \leq c$ changes, and let $r := c - c'$ be the number of remaining changes that must be placed outside this window.

This middle window acts as a separator between the left and right parts of the sequence. Since each seed has span w, no seed can simultaneously depend on positions strictly to the left and strictly to the right of this window. Therefore, changes placed on the right side of the middle window cannot affect hits located on the left side, and changes placed on the left side cannot affect hits located on the right side. Hence, after fixing the configuration of changes Γ inside the middle window, the two sides become independent subproblems.

Let ℓ_{left} and ℓ_{right} denote the lengths of the left and right parts of the sequence, respectively, such that $n = \ell_{\text{left}} + w + \ell_{\text{right}}$. The position of the middle window can be chosen such that the two sides are almost balanced, i.e., $|\ell_{\text{left}} - \ell_{\text{right}}| \leq 1$. Therefore, it is sufficient to compute the DP only up to length $\max\{\ell_{\text{left}}, \ell_{\text{right}}\} = \lceil (n - w)/2 \rceil$.

For each fixed placement of the c' changes inside the middle window, the remaining r changes are distributed between the two independent parts. More precisely, for each $r' = 0, 1, \ldots, r$, we place r' changes on the left side and $r - r'$ changes on the right side. The corresponding MinHits value is then obtained by combining the DP values of the two subproblems. Finally, the minimum over all possible splits r', and over all possible placements of the c' changes inside the middle window, gives the desired value for this case.

Reducing the number of changes in history sets by 1. In the minimizing DPs above, the maximal number of changes to distribute across the length-n sequence is c, and in principle, they may all be in the same history window of length w (MinHits) or $2w - 1$ (MinCov, MinCov|Hits). We can special-case this particular situation and reduce the general DP from considering c changes to only $c - 1$ changes, reducing the number of histories Γ or Δ considerably.

If a particular history window contains all c changes, then no additional changes remain to be placed elsewhere in the sequence. In the MinHits DP (Section 3.2), the update case 2 (new change) cannot happen anymore, and instead of looping through case 1 (no new change) until $i = n$, we pre-compute and tabulate the resulting MinHits values for all Γ with $|\Gamma| = c$. Consequently, the resulting MinHits or MinCov value can be computed directly, without extending the dynamic programming recursion beyond this point.

We omit the technical details for MinCov and MinCov|Hits at this point.

▷ Claim 6. The modified MinHits DP algorithm needs time $O(nc \cdot \binom{w}{\leq c-1} + w\binom{w}{c})$ and space $O(c \cdot \binom{w}{\leq c})$.

Proof. This is a direct consequence of Claim 2 and the $c \mapsto c-1$ reduction. The $O(w\binom{w}{c})$ time term is for pre-computing the lookup table of size $\binom{w}{c}$. ◁

3.6.2 Asymptotic improvement: Limited changes per window

The minimum number of hits (or coverage) is typically not attained when all c changes are concentrated within a one or a few history windows of length w (or $2w-1$), particularly if $n \gg w$. In such cases, equivalent or better solutions usually exist in which the number of changes within any window of length w is bounded by a value $u \leq c$. Under this assumption, the sets Γ considered in the dynamic programming algorithm can be restricted to subsets of size at most u, rather than at most c. Consequently, the number of possible histories Γ for the MinHits DP is reduced from $\binom{w}{\leq c}$ to $\binom{w}{\leq u}$, which may lead to a substantial reduction in both running time and memory usage. An analogous argument applies to the MinCov DPs.

The key question is how to find a safe upper bound u that does guarantee to find the correct optimal MinHits value. We derive such a bound by analyzing the range over which a set of changes can affect seed hits.

Consider an infinite sequence $\ldots, s_{-1}, s_0, s_1, s_2, \ldots$ and a window $s_i, s_{i+1}, \ldots, s_{i+w-1}$ of length w. Any change within this window can only affect seeds that overlap it. The earliest affected seed starts at position $i - w + 1$, and the latest affected seed starts at position $i + w - 1$. Therefore, all affected seeds are contained within the interval $s_{i-w+1}, \ldots, s_{i+2w-2}$, which has length $3w - 2$.

Now let u' denote the minimum number of changes required to destroy all hits in any sequence of length $3w - 1$. Suppose that a window of length w contains u' or more changes. Since these changes influence only a region of length at most $3w - 2$, but the u' changes are sufficient to eliminate all hits in a region of size $3w - 1$, at least one of the changes inside the window is unnecessary and can be moved elsewhere (or omitted) without increasing the number of hits. It follows that there always exists an optimal solution in which every window of length w contains at most $u := u' - 1$ changes.

This bound is useful when $u < c$, which may occur when the total number of changes is large. In the experiments presented in this paper, the number of changes is small, and thus this optimization has limited practical impact. Nevertheless, it provides an improvement and may be valuable for future studies involving larger values of c.

4 Experimental Results

4.1 Comparison of optimization times

We compare running times of the ILP and DP approach for the MinHits problem. Timings were run on an Ubuntu (24.04 LTS) server with two AMD EPYC 9534 64-Core Processors, 1.5 TB of 4800-MHz DDR5 memory, and a KIOXIA CD8P SSD. For ILPs, we use a free

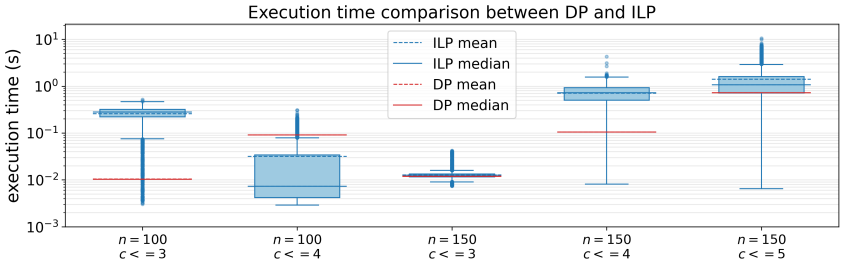

Figure 3 Running times (logarithmic scale) for the ILP and DP approaches (without optimizations) for the MinHits objective on all masks of shape $(25, 35)$ with different sequence lengths n and numbers of changes c. The blue boxplots and lines represent the distribution of ILP execution times over masks; the red lines indicate the corresponding (mask-independent) DP time statistics.

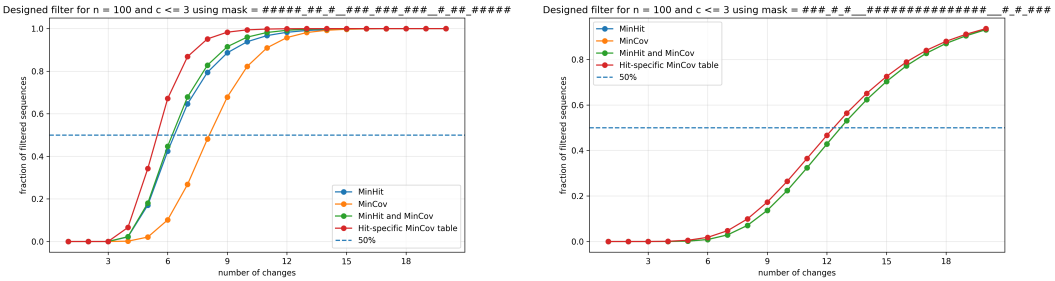

Figure 4 Filtration efficiency curves for two masks of shape (25,35), with filtration criteria F designed for $n = 100$ and $c^* = 3$ changes, for the four filter types introduced in Section 3.

academic Gurobi v13 license [7] and the Python API. Due to space constraints, we omit results for MinCov and MinCov|Hits, which are structurally similar but much longer overall. Figure 3 shows boxplots of the distribution of running times across all symmetric $(25, 35)$-shaped masks for different combinations of $n \in \{100, 150\}$ and $c \in \{3, 4, 5\}$. Since the DP timings are independent of the mask (the same work is done for each mask) and times may vary only due to possible uneven CPU or memory load on the server, only summary statistics are shown for DP. We observe that the ILP becomes faster for $n = 100$ when increasing c from 3 to 4. This happens because for $c = 4$, there are many masks with MinHits=0 objective value, which is easy to find even by the LP relaxation. The DP running times increase exponentially with c, as expected. They appear to be independent of $n \in \{100, 150\}$, which is explained by the setup cost (table allocation, pre-computation) that is independent of n. For $n = 150$, the DP approach has an advantage over the ILP solver, which would become more pronounced when we increase n and apply the proposed DP optimizations.

4.2 Comparison of masks

With both ILP and DP algorithms for MinHits, MinCov and MinCov|Hits, we can define the numerical criteria F for the 4 filter types mentioned in Section 3 for any mask of interest, and using enumeration or sampling for computing $\text{Filtered}_F(\kappa, n, c)$ we can compute or approximate filtration efficiency curves.

As exemplary results, Figure 4 shows the four curves for the $(k, w) = (25, 35)$ masks `#####_##_#__###_###_###__#_##_#####` (left) and `###_#_#___###############___#_#_###` (right), with filtration criteria F designed for $n = 100$ and $c^* = 3$ changes. Unless one

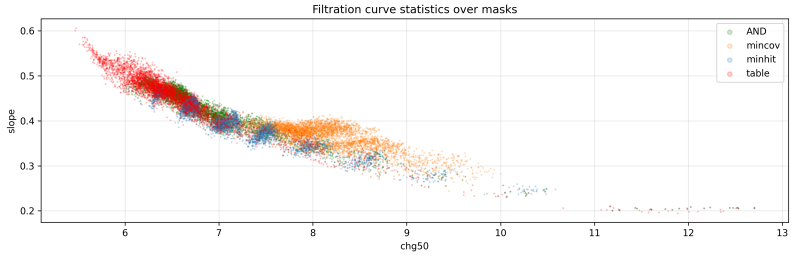

Figure 5 Comparison of chg50 values and slopes of all four filter types (color) of all 4 368 symmetric masks of shape $(25, 35)$. Top left is best; bottom right is worst.

has different goals, one considers an exact filter F better than another exact filter F' if the curve of F is above the curve of F', because then F filters out more of the undesired sequences with $c > c^*$ changes. (Both curves are at zero for $c \leq c^*$ because both F and F' are exact.) In the left panel of Figure 4, the MinCov filter (orange) performs worst, the MinHits (blue) filter better and close to the combined AND filter (green), which means that a simple logical AND of the hits and coverage criteria only yields a minor improvement, while the MinCov table filter (red; using MinCov|Hits, see Section 3) performs considerably better. This behavior is typical for many masks. However, for the mask in the right panel, the filtration curves for the MinHits, MinCov and combined AND filter agree (only the AND curve is therefore visible), the Table filter is only slightly better, and all four filters of this mask perform much worse than all filters of the left panel's mask.

We now introduce two quantities that capture the shape of the filtration curve and that may be used to compare the quality of (mask, filter) pairs designed for specific n and c^*:

- the *chg50* value is defined as the number of changes where the (interpolated) curve crosses 50% filtration rate;
- the *slope* is defined as $\max_{c=0,1,2,\ldots}(f(c+2) - f(c))$, where f is the filtration efficiency curve, the largest jump in filtration rate when the number of changes increases by 2. (The chosen difference of 2 is somewhat arbitrary, but better resolves between curves than a difference of 1.)

Figure 5 visually compares these two properties for all symmetric masks of shape $(25, 35)$. Several interesting observations may be made. The Table filters show the best filtration properties. The (red) point with largest slope (> 0.6) corresponds to the mask in the left panel of Figure 4; this can therefore considered to be the best mask. There is a number of badly performing masks (bottom right corner); and the mask in the right panel of Figure 4 is an example of them. On a side note, the MinHits filter chg50 values come in clusters on the chg50 axis, likely due to the discreteness of the low 1-digit MinHits numbers.

5 Discussion

We have made novel contributions to the field of spaced seed design: For the first time, we have designed a filtration strategy that combines hit-based and coverage-based criteria beyond a simple logical AND, and we have shown the table-based MinCov|Hits strategy to be quite effective for most masks. To compute the MinCov|Hits table, we have given both an ILP and a DP formulation that builds on simpler (but structurally similar) DPs for the related MinHits and MinCov optimization problems. The DP formulations reduce the number of change configurations to be considered from $\binom{n}{\leq c}$ to $\binom{w}{\leq c}$, independently of n. We have furthermore pointed out potential optimizations for reducing the running time of the

DP approaches. Implementing them and examining their effect for longer sequences is part of ongoing work. For the small number of changes that are of interest for short sequences ($n = 100$), we found them not to be relevant. However, the number of changes of interest grows with $\Theta(n)$ as the sequence length grows, and reducing the numbers of history sets from $\binom{w}{\leq c} = 2^w$ for $c \geq w$ to $\binom{w}{\leq u}$ for a small value of $u < w$ that does not depend on c becomes crucial.

While we found the exact computation of filtration efficiency curves (for large w or c) to be impractical, we analyzed the curves for all $(25, 35)$-shaped symmetric masks within a small case study by random sampling up to three digits accuracy. We defined two characterizing quantities of a mask/filter pair (κ, F) designed to be exact for a given sequence length n and number of changes c^*: the chg50 value and the slope of the filtration curve. We observed large differences between masks and found the optimal symmetric $(25, 35)$ mask for $n = 100$ and $c^* = 3$ to be `#####_##_#__###_###_###__#_##_#####`.

In ongoing and future work, we intend to build an atlas for spaced seeds with their hit- and coverage-related properties and filtration efficiencies, examine alternative bounds for limiting the number of changes per history window in the DP approaches, and extend the ideas presented here to combinations of masks. In addition, where this work only provides theoretical insights, the practical consequences of using an optimal vs. an average mask should be quantified in terms of wall-clock time speedup. Naturally, such an evaluation would depend on many additional design decisions and parameters.

We also need to point out a practical limitation of the approach: The MinCov|Hits thresholds assume that no additional random hits occur; otherwise, the filter may lose its exactness property. In practice, this may happen, depending on how the filter is used in a concrete application. For example, assume that all κ-mers of the human genome are stored in an index and the κ-mers of a sequenced DNA read of length $n = 100$ are queried. Assume further that the read is indeed a substring of a (unique) location in the genome with $c = 3$ substitutions that occur at certain positions of the read. This defines a hit pattern and coverage pattern on the read, but additionally, some of the changed κ-mers may occur *elsewhere* in the genome, yielding additional hits and additional covered positions, if the index only stores the presence of the κ-mer in the genome, but not its position(s). A problem can occur if the resulting hit and coverage pattern are not among the possible $\binom{n}{\leq c}$ patterns for at most c changes; the observed coverage may then be *below* the MinCov|Hits value of the increased hit value because it was not considered in the optimization. Here is an example with a $(25, 35)$-mask, $c = 3$, $n = 100$, and 12 hits (with MinCov|Hits 77) plus one additional random hit, yielding a coverage of 77 plus 1, while the MinCov|Hits for 13 hits from 3 changes is 79:

```
mask:      #####_##_#__###_###_###__#_##_#####  (shape (25, 35))
3 changes: 00000000000000000000000000000000001000000000000000000000000000011000000000000000000000000000000000
12+1 hits: 0000010010110+0100010001101001000000000001000000000000100000000000  (+ is an additional random hit)
77+1 cov:  00000111111111111111111111111111111+1111111111111111111111111111100111111111111001011011111000000000000
```

The problem can be completely avoided by computing safe MinCov thresholds that are valid for *all* hit distributions with h hits, but this reduces filtration efficiency.

Code is available at `https://gitlab.com/rahmannlab/seed-optimization`.

References

1 B. Brejová, D. G. Brown, and T. Vinar. Optimal spaced seeds for homologous coding regions. *J Bioinform Comput Biol*, 1(4):595–610, January 2004. doi:10.1142/S0219720004000326.

2 Karel Brinda. Languages of lossless seeds. In Zoltán Ésik and Zoltán Fülöp, editors, *Proceedings 14th International Conference on Automata and Formal Languages, AFL 2014, Szeged, Hungary, May 27-29, 2014*, volume 151 of *EPTCS*, pages 139–150, 2014. doi:10.4204/EPTCS.151.9.

3 Stefan Burkhardt, Andreas Crauser, Paolo Ferragina, Hans-Peter Lenhof, Eric Rivals, and Martin Vingron. q-gram based database searching using a suffix array (QUASAR). In Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Proceedings of the Third Annual International Conference on Research in Computational Molecular Biology, RECOMB 1999, Lyon, France, April 11-14, 1999*, pages 77–83. ACM, 1999. doi:10.1145/299432.299460.

4 Stefan Burkhardt and Juha Kärkkäinen. Better filtering with gapped q-grams. *Fundam. Informaticae*, 56(1-2):51–70, 2003. URL: http://content.iospress.com/articles/fundamenta-informaticae/fi56-1-2-04.

5 K. Břinda, M. Sykulski, and G. Kucherov. Spaced seeds improve k-mer-based metagenomic classification. *Bioinformatics*, 31(22):3584–3592, November 2015. doi:10.1093/BIOINFORMATICS/BTV419.

6 M. C. Frith and L. Noé. Improved search heuristics find 20,000 new alignments between human and mouse genomes. *Nucleic Acids Res*, 42(7):e59, April 2014.

7 Gurobi Optimization, LLC. Gurobi Optimizer reference manual, 2023. URL: https://www.gurobi.com.

8 L. Hahn, C. A. Leimeister, R. Ounit, S. Lonardi, and B. Morgenstern. rasbhari: Optimizing spaced seeds for database searching, read mapping and alignment-free sequence comparison. *PLoS Comput Biol*, 12(10):e1005107, October 2016. doi:10.1371/JOURNAL.PCBI.1005107.

9 Moein Karami, Jens Zentgraf, and Sven Rahmann. Spaced Seed Optimization. Software, version 0.11., swhId: swh:1:dir:050c8caccda68021f2feb9b7b8400444e9888340 (visited on 2026-08-14). URL: https://gitlab.com/rahmannlab/seed-optimization/, doi:10.4230/artifacts.27677.

10 Denise Y. F. Mak and Gary Benson. All hits all the time: parameter-free calculation of spaced seed sensitivity. *Bioinformics*, 25(3):302–308, 2009. doi:10.1093/BIOINFORMATICS/BTN643.

11 François Nicolas and Eric Rivals. Hardness of optimal spaced seed design. *J. Comput. Syst. Sci.*, 74(5):831–849, 2008. doi:10.1016/J.JCSS.2007.10.001.

12 Laurent Noé and Donald E. K. Martin. A coverage criterion for spaced seeds and its applications to support vector machine string kernels and k-mer distances. *J. Comput. Biol.*, 21(12):947–963, 2014. doi:10.1089/CMB.2014.0173.

13 L. Noé. Best hits of 11110110111: model-free selection and parameter-free sensitivity calculation of spaced seeds. *Algorithms Mol Biol*, 12:1, 2017.

14 Kim R. Rasmussen, Jens Stoye, and Eugene W. Myers. Efficient q-gram filters for finding all epsilon-matches over a given length. In Satoru Miyano, Jill P. Mesirov, Simon Kasif, Sorin Istrail, Pavel A. Pevzner, and Michael S. Waterman, editors, *Research in Computational Molecular Biology, 9th Annual International Conference, RECOMB 2005, Cambridge, MA, USA, May 14-18, 2005, Proceedings*, Lecture Notes in Computer Science, pages 189–203. Springer, 2005. doi:10.1007/11415770_15.

15 Jens Zentgraf and Sven Rahmann. Design of worst-case-optimal spaced seeds. In Brona Brejová and Rob Patro, editors, *25th International Conference on Algorithms for Bioinformatics, WABI 2025, University of Maryland, College Park, MD, USA, August 20-22, 2025*, LIPIcs, pages 22:1–22:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPIcs.WABI.2025.22.

16 L. Zhang. Superiority of spaced seeds for homology search. *IEEE/ACM Trans Comput Biol Bioinform*, 4(3):496–505, 2007. doi:10.1109/TCBB.2007.1013.